LiFS: An Attribute-Rich File System for Storage Class Memories

Sasha Ames Nikhil Bobb Scott A. Brandt Carlos Maltzahn Ethan L. Miller Kevin Greenan Owen Hofmann Mark Storer

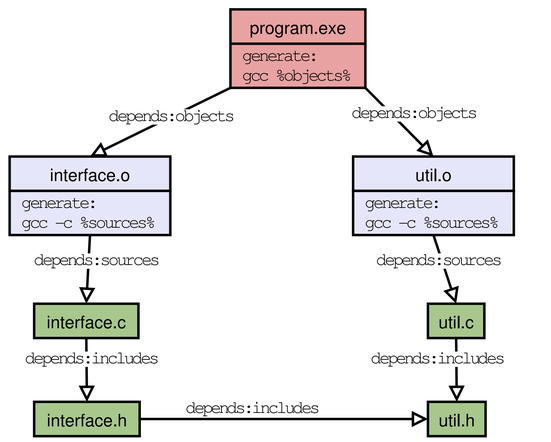
Storage Systems Research Center
Computer Science Department
University of California, Santa Cruz

Motivation

- The amount and variety of data stored has grown much faster than existing file systems.
- Many applications keep their own metadata stores, e.g. iTunes, e-mail clients, etc. These systems:
  - are application-specific
  - add significant complexity
- It is often easier to search the web than the local file system
  - Web search engines such as Google can take advantage of a rich hyperlink structure.
  - Similar contextual data in the filesystem would enhance local file searches.

LiFS

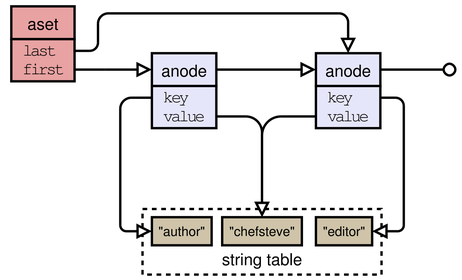
- Incorporating rich metadata into the filesystem itself simplifies sharing between applications.
- Applications need only create metadata, not store it.
- LiFS (The Linking FileSystem) introduces the relational link as a unit of file metadata.
  - Link files with $\langle key, value \rangle$ pairs expressing their relationship.
  - Traditional directories are 0-byte files with outgoing links.



A compiler dependency graph represented with relational links

Design

- Designed to use non-volatile memory for metadata storage
  - Fast storage-class memories provide speed necessary for searching rich metadata structures
  - Random access memories allow for simpler, efficient code
    - Linked lists store objects rather than trying to lay out structures on a disk block
    - A string table saves space and allows for easy string comparisons



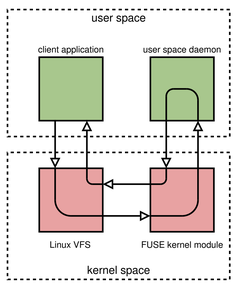
A set of attributes from a link or file

- Several new system calls have been introduced to manipulate relational links

System call	Function
rellink	Create a new relational link between files
rmlink	Remove a relational link between files
setlinkattr	Set attributes on an existing link between files
openlinkset	Returns an identifier for a set of links from a source file
readlinkset	Fills in standard directory entry structure with link name and attributes for the next link in a set

Implementation

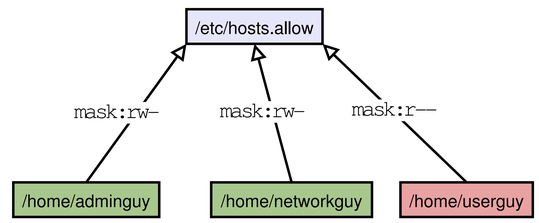
- LiFS was implemented through FUSE (Filesystem in Userspace)
  - Prototype LiFS without complexity of in-kernel development
  - Even with significant performance overhead of FUSE, LiFS' performance is competitive with existing kernelspace filesystems



Future Work

- In-kernel implementation of LiFS
  - Allows for better comparison with existing filesystems
  - Because FUSE is designed around existing POSIX interfaces, adding new functionality is difficult.
- Take advantage of relational links in file searches
  - What kinds of new queries can users take advantage of?
  - How should searching be presented to the user?
    - Existing approaches often use directory components as search terms, e.g. `/search/owner:john/type:mp3/`

- Can LiFS metadata be stored with other storage technologies?
  - LiFS on disk
    - Seek time of disk accesses makes searching link structures difficult
    - A number of graph indexing solutions exist which might make LiFS on disk possible
  - LiFS in flash
    - Different flash technologies have different access properties
    - Flash has a limited number of writes per block
    - To change data, an entire block must be erased and rewritten
  - LiFS implemented using extended attributes
    - System would be backwards-compatible
    - Standard xattr interface is inefficient



An ACL implemented using relational links

- Many different paths can exist to a single file
- Allowing path to modify file access is a powerful tool
  - Security
    - Providing users links to files might replace complex ACLs
  - Modifying file data
    - Individual links might serve to compress/encrypt file data
    - Files might be defined as an aggregation of linked files